2011 HEC-FSIO Workshop
Measurement and Understanding Panel

Accessible and Reliable Parallel I/O Benchmarking

Xiaosong Ma

North Carolina State University
Joint faculty with Oak Ridge National Lab

Benchmarks Needed

- Benchmarks widely used
  - Processor, compiler, OS: SPEC
  - Parallel computing: LinPack, NAS
  - Database and storage: TPC, SPC, IO-Zone

- Uses in HEC I/O R&D
  - HPC application developers and users: selecting libraries, I/O modes, …
  - Supercomputer, parallel file system, or I/O library designers: evaluating/validating design and implementation
  - HEC system owners: hardware provisioning and configuration

More helpful with Exascale on its way

State of the Art

- Parallel I/O benchmarks exist

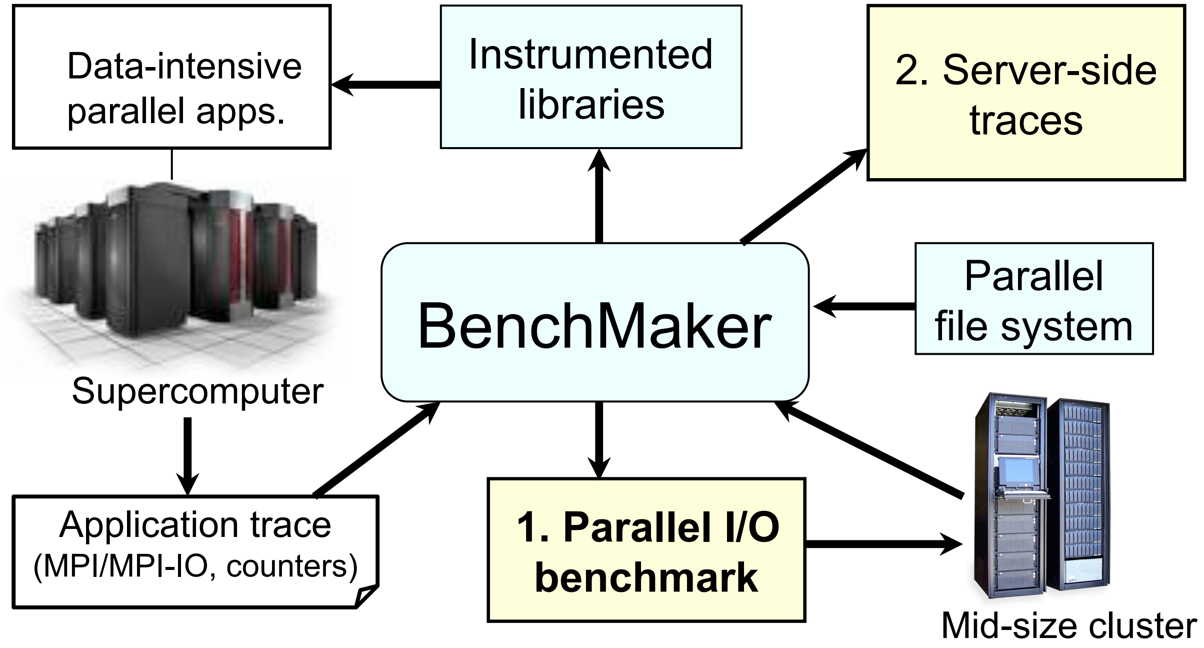
In scalable I/O and file systems, there are few generally applicable tools available. Tools and benchmarks for use by application programmers, library developers, and file system managers would be of enormous use for the future.

-Multiagency FSIO Suggested R&D Topics
2005-2009

"Measurement and understanding of system workload in HEC environment" rated "very important" in HEC-FSIO Roadmap 2010

Our Ongoing HECURA Project

- BenchMaker: automated extraction of parallel I/O benchmarks from HPC applications
  - Faithfully re-creates I/O related activities
  - Smaller, portable, and shorter runs
  - Human intelligible
  - Grows with applications



Data-intensive parallel apps.

Instrumented libraries

2. Server-side traces

Supercomputer

BenchMaker

Parallel file system

Application trace (MPI/MPI-IO, counters)

1. Parallel I/O benchmark

Mid-size cluster

⌖Joint work at NCSU, U. of Rochester, and UIUC

Client-side Benchmark Generation

```
Main() {
 Read_input(){ …
  MPI_File_read(); …}
 for (i=0; i<MAX_STEP; i++) {
  solve(); //comp. phase
  update(); {//comm. Phase
   MPI_Bcast(); …}
  write_output(i, …) { …
   MPI_File_write_all();
   …}
 }
 finalize();
}
```

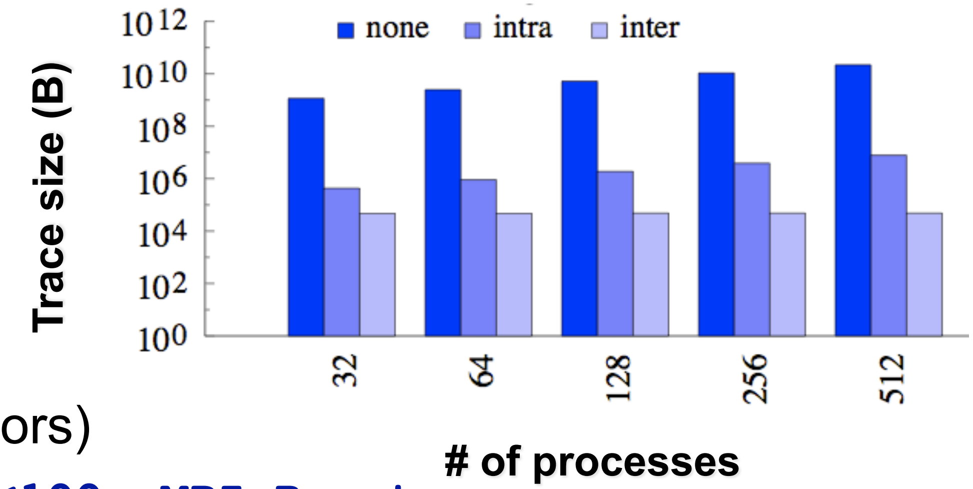
Original application

```
Main() {
 initialize'();
 MPI_File_read();
 for (i=0; i<MAX_STEP; i++) {
  compute'();
  communicate'();

  MPI_File_write_all();
 }


 finalize'();
}
```

BenchMaker generated benchmark

Benchmark Generation Progress

- ✓ ScalaIOTrace
  - On-the-fly I/O trace compression [PDSW09]
  - MPI-IO and POSIX
  - PRSD (Power Regular Section Descriptors)
    - Loops -> PRSDs (e.g., **`<100, MPI_Bcast, MPI_File_write_all>`**)



- ✓ Histogram-based trace collection and replay
  - Retrieving and compressing statistics on event participation and timing data
  - Deadlock free replay
- Code generation
  - ➡ Revised trace replayer
  - PRSDs back to loops

Benchmark Generation by Slicing

- Alternative approach to trace-based benchmark extraction
  - Based on dependence analysis
  - Code-to-code benchmark generation possible
- Justification
  - Payload content unimportant for I/O benchmarks
  - Key HPC I/O operation parameters not involved in main computation steps
  - Potential “plan A”
- Current progress (in collaboration with Tsinghua University, China)
  - ✓ Able to slice MPI-IO calls from applications
  - ➡ Intermediate to source code transformation
  - ➡ Adding POSIX and netCDF support

Next Step: Establishing Base Application Set

- Applications included in plan
  - CCSM, POP, Grapes
  - GTC, S3D, FLASH
  - mpiBLAST

- How complete and representative is the set?
  - Domains
    - Numerical simulations, biological database search, graph,? visualization?
  - I/O behavior
    - File access: shared, per-process file
    - I/O libraries: MPI-IO, netCDF/HDF5, ADIOS, POSIX
    - I/O operations: write-intensive, read-intensive?
    - I/O patterns: sequential, random?
  - Others? Suggestion welcome!

Reliable and Reproducible I/O Performance Measurement

- Can we trust numbers obtained on supercomputers?
- Shared nature of large-scale HEC platforms
  - Multiple jobs execute simultaneously, plus interactive accesses
  - I/O system shared between co-running jobs
  - Dedicated access to entire machine nearly impossible
  - Result: huge variance in observed I/O performance

S3D Job Scale	600 MPI Processes		30000 MPI Processes	
Trial Number	Time Taken (s)	Total IO Rate (MB/s)	Time Taken (s)	Total IO Rate (MB/s)
1	0.649	3120.185	315.078	321.349
2	46.753	43.313	64.164	1577.988
3	35.816	56.539	46.868	2160.323

Sample I/O performance measurement from multiple S3D runs on Jaguar at ORNL

Situation Worse with Exa-scale Machines

- Million-way parallelism, higher pressure on shared storage systems
  - Example
    - Large number of concurrent metadata operations
    - Max time measured with 1024 process concurrently opening files on Jaguar: varying from 0.64 to 167 seconds!

- Larger gap in scale between production systems and development clusters
  - Private, local clusters less capable of demonstrating I/O performance for production runs

Ongoing Research on I/O Performance Variance Control

- Performance noise removal
  - Extracting application’s internal behavior from multiple unreliable trial measurements?
    - Challenging due to lack of references
    - May be resource heavy (many runs needed)
- Devising approaches for performance variance reduction
  - Most large-scale parallel applications perform I/O periodically
    - Typical SciDAC apps: one checkpoint per hour, result data output frequency ranging from per 10 minutes to per hour
  - Possible to coordinate multiple applications’ periodic I/O activities through I/O middleware?

Thank You!

Regenerating What’s Between I/O Phases

- Computation and communication needed
  - But only enough to retain impact on I/O
    - Memory footprint and access pattern
    - I/O intensiveness and overlap behavior
- Creating realistic computation/communication workload
  - Combining micro computation kernels
  - Memory behavior tracing
- Benchmark users allowed to configure levels of computation contents
  - High, low, none …

Making Benchmarks Parametric

- Adjustable computation modes
  - Problem size
  - Number of processes

- Adjustable I/O modes
  - I/O grouping
  - Buffering options (memory and local storage)
  - On-disk data layout
  - Dedicated I/O processor/core per node
  - I/O frequency

Evaluation and Suite Composition

- Benchmark fidelity evaluation
  - Goal: ressembling base application
- Building benchmark suite
  - Goal: completeness and non-redundancy

- Common challenge: how to quantify difference between executions

Server-side Trace Generation

- Dilemma for server-side designers
  - Large machines do not allow privileged accesses
    - Or trace collection
  - Small clusters do not have enough nodes as clients
- Creating server-side traces from benchmarks
  - Translate 10,000-client benchmark into 100-server traces
  - Extrapolating client->server mappings from mid-sized runs

Dissemination and Acknowledgment

- Plan to release
  - Open source tool
  - Prototype benchmarks
- Call for community participation
  - Requirement from FSIO researchers
  - Applications
  - Benchmark test users
- We appreciate support and input from
  - NSF
  - Rajeev Thakur and Rob Ross (ANL), Scott Klasky, Phil Roth, and Galen Shipman (ORNL)